Introduction to Singularity: A Container Platform for Scientific and High-Performance Computing

Marty Kandes, Ph.D.

Computational & Data Science Research Specialist
High-Performance Computing User Services Group
San Diego Supercomputer Center
University of California, San Diego

XSEDE Webinar
Thursday, January 21st, 2021
11:00 AM - 12:00 PM PST
XSEDE Code of Conduct

https://www.xsede.org/codeofconduct
Today

- What is a container?
- What is Singularity?
- What can Singularity do for you?
Tomorrowland

- Introduction to Singularity (Today)
- Creating Containerized Workflows with Singularity (SI21)
- Advanced Singularity (TBD)
SDSC Summer Institute

The San Diego Supercomputer Center Summer Institute is a week-long workshop held at the University of California, San Diego that focuses on a broad spectrum of introductory-to-intermediate topics in High Performance Computing and Data Science. The purpose of the Summer Institute is to give the attendees an overview of topics in High Performance Computing and Data Science and accelerate their learning process through highly interactive classes with hands-on tutorials on the Comet Supercomputer. The program is aimed at researchers in academia and industry, especially in domains not traditionally engaged in supercomputing, who have problems that cannot typically be solved using local computing resources.

This year’s Summer Institute continues SDSC’s strategy of bringing High Performance Computing to the Long Tail of Science, i.e. providing resources to a larger number of modest-sized computational research projects that represent, in aggregate, a tremendous amount of scientific progress.

Past Summer Institutes

- 2020: High Performance Computing and Data Science
- 2019: High Performance Computing and Data Science

https://www.sdsc.edu/education_and_training
Q: What can hold 10,000 iPads ...

... and travel from Shanghai to Hamburg for $500?
A: Shipping Container

Est. 1956
Before Containerization (Bulk Break Cargo)

- Bespoke - goods must be loaded and unloaded individually, many times by hand
- Inefficient - more time spent loading and unloading goods than transporting them
- Insecure - goods must be stored and handled by intermediaries during transport; potential loss and/or theft of goods
- Local - only luxury and specialty goods shipped long-distance
After Containerization (Intermodal Freight Transport)

- Standardized - containers of known dimensions and permissible weight tolerances
- Efficient - portable containers allow fast loading and unloading from multiple modes of transportation
- Secure - goods remained stored within the same container during transport
- Global - cost effective to ship almost any good from anywhere in the world
Computing ≈ Shipping?
Evolution of Software Deployment
Software Deployment on XSEDE
Comet vs. Stampede2
What is a (Software) Container?

A (software) container is an abstraction for a set of technologies that aim to solve the problem of how to get software to run reliably when moved from one computing environment to another.
A container image is simply a file (or collection of files) saved on disk that stores everything you need to run a target application or applications: code, runtime, system tools, libraries, etc.
Container Process

A container process is simply a standard (Linux) process running on top of the underlying host’s operating system and kernel, but whose software environment is defined by the contents of the container image.
A container process is simply a standard (Linux) process running on top of the underlying host’s operating system and kernel, but whose software environment is defined by the contents of the container image.
A container process is simply a standard (Linux) process running on top of the underlying host’s operating system and kernel, but whose software environment is defined by the contents of the container image.
“... it’s our loading program.”

“We can load anything ... anything we need.”
Containers vs. Virtual Machines

Container-based applications have *direct access* to the host kernel and hardware and, thus, are able to achieve similar performance to native applications. In contrast, VM-based applications only have *indirect access* via the guest OS and hypervisor, which creates a significant performance overhead.
Advantages of Containers

▶ **Performance**: Near-native application performance

▶ **Freedom**: Bring your own software environment

▶ **Reproducibility**: Package complex software applications into easy to manage, verifiable software units

▶ **Compatibility**: Built on open standards available in all major Linux distributions

▶ **Portability**: Build once, run (almost) anywhere
Limitations of Containers

- **Architecture-dependent**: Always limited by CPU architecture (x86_64, ARM) and binary format (ELF)
- **Portability**: Requires glibc and kernel compatibility between host and container; also requires any other kernel-user space API compatibility (e.g., OFED/IB, NVIDIA/GPUs)
- **Filesystem isolation**: filesystem paths are (mostly) different when viewed inside and outside container
Docker

- Most common container platform in use today
- Provides tools and utilities to create, maintain, distribute, and run containers images
- Designed to accommodate network-centric services (web servers, databases, etc)
- Easy to install, well-documented, and large, well-developed user community and container ecosystem (DockerHub)

https://www.docker.com
Docker on HPC Systems

- HPC systems are shared resources
- Docker’s security model is designed to support trusted users running trusted containers; e.g., users can escalate to root
- Docker not designed to support batch-based workflows
- Docker not designed to support tightly-coupled, highly distributed parallel applications (MPI).
Singularity: A Container Platform for HPC

- Reproducible, portable, sharable, and distributable containers
- No trust security model: untrusted users running untrusted containers
- Support HPC hardware and scientific applications

https://github.com/hpcng/singularity
https://www.sylabs.io
Features of Singularity

▶ Each container is a single image file
▶ No root owned daemon processes
▶ No user contextual changes or root escalation allowed; user inside container is always the same user who started the container
▶ Supports shared/multi-tenant resource environments
▶ Supports HPC hardware: Infiniband, GPUs
▶ Supports HPC applications: MPI
Most Common Singularity Use Cases

- Building and running applications that require newer system libraries than are available on host system
- Running commercial applications binaries that have specific OS requirements not met by host system
- Converting Docker containers to Singularity containers
The Singularity Workflow

1. **Build** your Singularity containers on a local system where you have root or sudo access; e.g., a personal computer where you have installed Singularity

2. **Transfer** your Singularity containers to the HPC system where you want to run them

3. **Run** your Singularity containers on that HPC system
Running Singularity on Mac OS X or Windows

1. Install VirtualBox on your personal computer:

2. Create either an Ubuntu or Fedora-based virtual machine, where you will build and test your Singularity containers

3. Install Singularity* on that virtual machine:

* Recommendation: Install the same version of Singularity used on the HPC system where you plan to run your containers. If you plan to run on multiple HPC systems, then install the lowest version number you expect to use.
The main Singularity command

```
singularity [options] <subcommand> [subcommand options] ...```

has three essential subcommands:

- **build**: Build your own container from scratch using a Singularity definition file; download and assemble any existing Singularity container; or convert your containers from one format to another (e.g., from Docker to Singularity)

- **shell**: Spawn an interactive shell session in your container.

- **exec**: Execute an arbitrary command within your container.
Building a Singularity Container

```
sudo singularity build ubuntu.sif ubuntu.def
```

```
mkandes@castlebravo:~$ ls
Desktop  Downloads  Dropbox  ubuntu.def
mkandes@castlebravo:~$ sudo singularity build ubuntu.simg ubuntu.def
Using container recipe deffile: ubuntu.def
Sanitizing environment
Adding base Singularity environment to container
I: Retrieving InRelease
I: Checking Release signature
I: Valid Release signature (key id 790BC7277767219C42C86F933B4FE6ACC0B21F32)
I: Retrieving Packages
I: Validating Packages
I: Resolving dependencies of required packages...
I: Resolving dependencies of base packages...
I: Found additional base dependencies: gcc-5-base gnupg gpgv libapt-pkg5.0 libblz
4-1 libreadline6 libstdc++6 libusb-0.1-4 readline-common ubuntu-keyring
I: Checking component main on http://us.archive.ubuntu.com/ubuntu...
I: Retrieving adduser 3.113+nmu3ubuntu4
I: Validating adduser 3.113+nmu3ubuntu4
I: Retrieving apt 1.2.10ubuntu1
I: Validating apt 1.2.10ubuntu1
I: Retrieving base-files 9.4ubuntu4
I: Validating base-files 9.4ubuntu4
I: Retrieving base-passwd 3.5.39
I: Validating base-passwd 3.5.39
```
A Singularity definition file is the starting point for designing any custom container.

It is a manifest of all software to be installed within the container, environment variables to be set, files to be added, directories to be mounted, container metadata, etc.

You can even write a help section, or define modular components in the container.
naked-singularity

- A repository of definition files for building Singularity containers around the software applications, frameworks, and libraries you need to run on high-performance computing systems.

- Aim of the project is to:
  1. Version control the Singularity containers we’re building, maintaining, and deploying for you;
  2. Make it easy for you to see what is installed within these Singularity containers; and
  3. Make available to you the same base definition files we use to build our Singularity containers, which can serve as a starting point for your own custom Singularity containers.

- [https://github.com/mkandes/naked-singularity](https://github.com/mkandes/naked-singularity)
Building a Singularity Container

```
sudo singularity build ubuntu.sif ubuntu.def
```
Interacting with a Singularity Container

```
Singularity ubuntu-docker.simg:~> which python
Singularity ubuntu-docker.simg:~> python --version
bash: python: command not found
Singularity ubuntu-docker.simg:~> which python3
Singularity ubuntu-docker.simg:~> python3 --version
bash: python3: command not found
Singularity ubuntu-docker.simg:~> exit
exit
mkandes@castlebravo:~$ 
```

```
singularity shell ubuntu-docker.sif
```
Running a Singularity Container

singularity exec ubuntu-docker.img python --version
MPI-based Singularity Containers

- Use the same Message Passing Interface (MPI) distribution and version within the container as would be used outside the container.

- If using Infiniband (IB), install the same OFED drivers and libraries inside the container as used on the underlying HPC hardware.
MPI-based Singularity Containers: MEEP

- MEEP: MIT Electromagnetic Equation Propagation is a free and open-source software package for simulating electromagnetic systems via the finite-difference time-domain (FDTD) method.

- Dependency hell: Too difficult to compile in most native software environments.
MPI-based Singularity Containers: MEEP

```bash
#SBATCH --job-name="meep-example"
#SBATCH --account=use300
#SBATCH --partition=debug
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=24
#SBATCH --time=00:30:00
#SBATCH --no-requeue
#SBATCH --output="meep-example.o%j.%N"

module purge
module load "$\{COMPILER_MODULE\}"
module load "$\{MPI_MODULE\}"
module load "$\{SINGULARITY_MODULE\}"
module list
printenv

time -p ibrunch singularity exec "$\{SINGULARITY_IMAGE_DIR\}/meep/meep.simg" meep /opt/meep/scheme/examples/parallel-wvgs-force.ctl

mpirun -np X singularity exec meep.sif meep parallel-wvgs-force.ctl
```
GPU-accelerated Singularity Containers

- GPU-accelerated containers also require an interface for accessing GPU drivers and libraries on the underlying host system.

- Traditionally, you would install the same driver and libraries within container that match distribution and version of them available on the host system.

- Today, Singularity actually allows you to bind mount the GPU driver and its supporting libraries at runtime with the \texttt{--nv} option.
GPU-accelerated Singularity Containers: TensorFlow

- TensorFlow is an open source software library for high performance numeric and symbolic computation, and is most popularly used today for machine learning applications such as neural networks.

- Like many of the most popular machine learning frameworks, TensorFlow continues to evolve rapidly, making it difficult to install on systems with older operating systems.
GPU-accelerated Singularity Containers: TensorFlow

singularity exec --nv keras-tensorflow-gpu.sif python mnist_deep.py
Singularity: A Summary

1. You can now install (almost) any software you like on your favorite HPC system without having to make a special request to the system’s administrators or user support staff.

2. In many cases, your software is now completely portable between the different HPC systems you want to run on.

3. And finally, you now have discrete software units (containers) that you can use to help maintain science reproducibility over the lifetime of a project, independent of how the software environment on any given HPC system changes over time.
Additional References

▶ Singularity User Guide:
  https://sylabs.io/guides/latest/user-guide/

▶ Example Singularity Definition Files:
  https://github.com/mkandes/naked-singularity

▶ Talks @ 1st Singularity User Group Meeting:
  https://sylabs.io/videos